

---

---

# S10/L5 Epicode Cybersecurity

— Malware analysis —

---

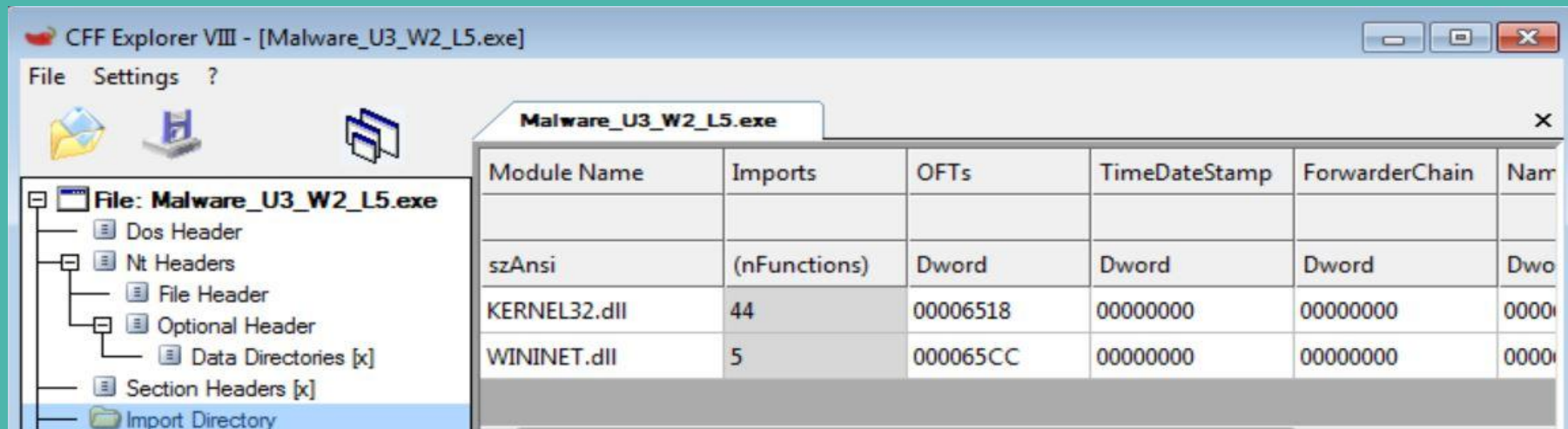
---

●	Introduzione	3
●	Analisi librerie malware	4
●	Analisi sezioni malware	5
●	Identificazione costrutti Assembly x86	6
●	Ipotesi funzionalità codice	8

Nell'esercizio di oggi andremo ad effettuare due attività fondamentali della malware analysis. Dato un file malware eseguibile, andremo a controllare le librerie importate e la sua composizione tramite CFF Explorer e, successivamente, analizzeremo un codice in linguaggio Assembly x86 per capire quali costrutti utilizza, ipotizzando quindi quale potrebbe essere la sua funzione.

Per prima cosa avviamo **CFF Explorer**, programma di analisi malware, ed andiamo a selezionare il file eseguibile di nostra competenza, “Malware\_U3\_W2\_L5.exe”. Per verificare le librerie che questo malware ha importato andiamo nella sezione ‘**Import Directory**’; possiamo vedere come il file in questione utilizzi due librerie:

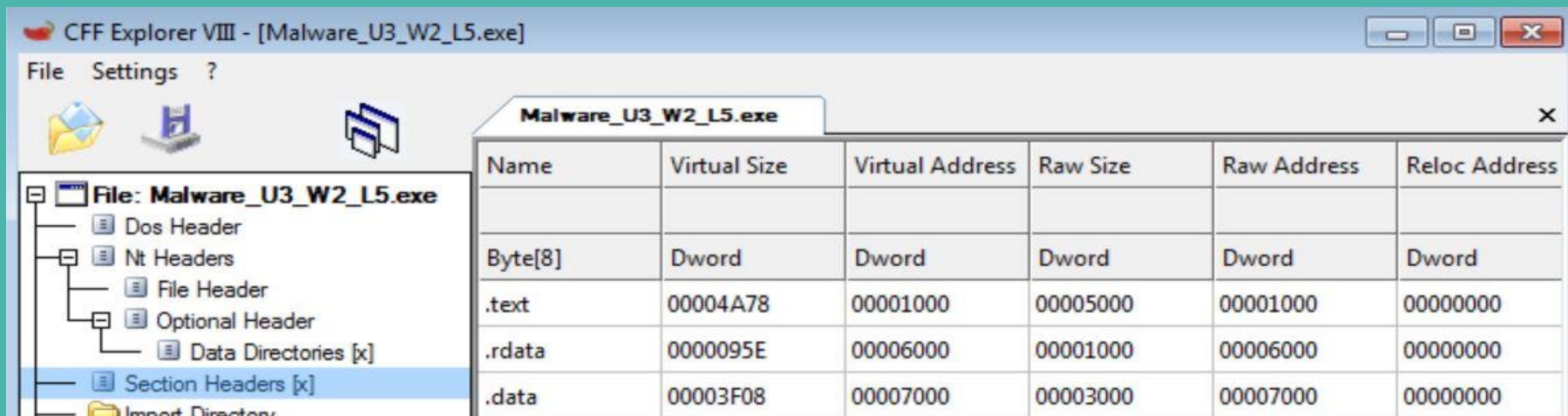
- **KERNEL32.dll**: libreria che contiene le funzioni principali di interazione col sistema operativo;
- **WININET.dll**: libreria che contiene le funzioni per l’implementazione di vari protocolli di rete.



Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name
szAnsi	(nFunctions)	Dword	Dword	Dword	Dwo
KERNEL32.dll	44	00006518	00000000	00000000	0000
WININET.dll	5	000065CC	00000000	00000000	0000

Andiamo ora nella sezione “**Section Headers**” per verificare le sezioni di cui il malware si compone. Ne troviamo tre:

- **.text**: sezione contenente le istruzioni che verranno eseguite dalla CPU all'avvio dell'eseguibile;
- **.rdata**: sezione che include le informazioni sulle librerie e le funzioni importate;
- **.data**: sezione contenente i dati e le variabili globali del file eseguibile.



CFF Explorer VIII - [Malware\_U3\_W2\_L5.exe]

File Settings ?

Malware\_U3\_W2\_L5.exe

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address
Byte[8]	Dword	Dword	Dword	Dword	Dword
.text	00004A78	00001000	00005000	00001000	00000000
.rdata	0000095E	00006000	00001000	00006000	00000000
.data	00003F08	00007000	00003000	00007000	00000000

Per la seconda parte dell'esercizio ci viene fornito un codice in linguaggio Assembly da analizzare per capire i costrutti da cui è composto e quale potrebbe essere la sua funzione. Già dalla prima riga di codice possiamo trovare il primo costrutto, ovvero la **creazione dello stack** tramite l'istruzione '**push**', seguito dal ciclo **if** che inizia dalle istruzioni '**cmp**' (compare) e '**jz**' (jump if zero). Alla fine del ciclo if vediamo l'ultimo costrutto utilizzato, ovvero l'**eliminazione dello stack** tramite il comando '**pop**'.

```
push    ebp
mov     ebp, esp
push    ecx
push    0          ; dwReserved
push    0          ; lpdwFlags
call    ds:InternetGetConnectedState
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jz      short loc_40102B
```

```
offset aSuccessInterne ; "Success: Internet Connection\n"
push    offset aSuccessInterne
call    sub_40117F
add     esp, 4
mov     eax, 1
jmp     short loc_40103A
```

```
loc_40102B:
push    offset aError1_1NoInte ; "Error 1.1: No Internet\n"
call    sub_40117F
add     esp, 4
xor     eax, eax
```

```
loc_40103A:
mov     esp, ebp
pop     ebp
retn
sub_401000 endp
```

In sintesi, andiamo ad elencare i costrutti:

- Creazione dello stack con il comando 'push';
- Ciclo if con i comando 'cmp' e 'jz';
- Eliminazione dello stack con 'pop'.

```
push    ebp
mov     ebp, esp
push    ecx
push    0           ; dwReserved
push    0           ; lpdwFlags
call    ds:InternetGetConnectedState
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jz      short loc_40102B
```

```
offset aSuccessInterne ; "Success: Internet Connection\n"
call    sub_40117F
add     esp, 4
mov     eax, 1
jmp     short loc_40103A
```

```
loc_40102B:
push    offset aError1_1NoInte ; "Error 1.1: No Internet\n"
call    sub_40117F
add     esp, 4
xor     eax, eax
```

```
loc_40103A:
mov     esp, ebp
pop     ebp
retn
sub_401000 endp
```

L'ultimo compito del progetto di oggi è quello di ipotizzare quale sia la funzionalità espletata dal codice in questione. Possiamo notare come, prima dell'inizio del ciclo if, in una stringa di codice sia presente una funzione **'calls:InternetGetConnectedState'**: questi due fattori ci fanno intuire che questo codice serva ad identificare la presenza della connessione ad internet della macchina su cui il codice viene eseguito. Tramite il ciclo if viene effettuato un controllo che ci porta due risultati, **'Success: Internet Connection'** o **'Error 1.1: No Internet'**, chiara evidenza dell'utilità di questa funzione.